UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Programmazione 3
e
Laboratorio di Programmazione 3

Java Database Connectivity

Proff. Angelo Ciaramella – Emanuel Di Nardo

Java Database Connectivity

- Java DataBase Connectivity (JDBC)
  - strato di astrazione software che permette alle applicazioni Java di connettersi a database
  - consente ad un'applicazione di accedere a diversi database senza dover essere modificata
    - devono supportare l'ANSI SQL 2 standard

- Componenti fondamentali di JDBC
  - un'implementazione del *vendor* del RDBMS (o di terze parti) conforme alle specifiche delle `API java.sql`
  - un'implementazione da parte dello sviluppatore dell'applicazione

Implementazione del vendor

- fornire un'implementazione di una serie di interfacce definite dal `package java.sql`
  - `Driver`
  - `Connection`
  - `Statement`
  - `PreparedStatement`
  - `CallableStatement`
  - `ResultSet`
  - `DatabaseMetaData`
  - `ResultSetMetaData`

- generalmente sono fornite le classi in un file `jar`

Implementazione del vendor

- Quattro tipologie di driver JDBC
  - JDBC-ODBC Bridge Driver
    - interfaccia JDBC verso il programma Java e un'interfaccia ODBC verso il database
  - Driver Native API Driver
    - è un driver scritto in un linguaggio nativo come il C
    - OCI Driver per Oracle
  - Network Protocol Driver (Middleware Driver)
    - è un driver scritto in Java installato su una macchina remota
  - Pure Java Driver
  - è un driver scritto in Java risiedente sulla stessa macchina del client
    - Thin Client di Oracle

Implementazione dello sviluppatore

- Un'applicazione JDBC deve
  - caricare un driver (non più necessario)
  - aprire una connessione con il database
  - creare un oggetto Statement per interrogare il database
  - interagire con il database
  - gestire i risultati ottenuti
  - **CHIUDERE** tutte le connessioni

Connessione SQLite

- Stringa di connessione
  - <protocol>:<database_name>
  - Ad ogni operazione la connessione va **aperta** e **chiusa**!

```
String protocol = "jdbc:sqlite:";

connect = DriverManager.getConnection(protocol +
"db.sqlite");

. . .

connect.close();
```

Connessione al database

Codice di riferimento

`JDBCApp.java`

Databese SQLite

- SQLite
  - Database compatto contenuto in un unico file
  - Supporta JDBC
  - Supportato dagli IDE
  - JDBC Driver:
    (https://github.com/xerial/sqlite-jdbc)

Codice di riferimento

`JDBCApp.java`

SQL Statements

- Permette di inviare istruzioni SQL al database
  - Statement
    - Operazione basilare
    - Utilizza tre metodi:
      - executeQuery() - SELECT
      - executeUpdate() – INSERT / UPDATE
      - execute() – Entrambi i casi
  - PreparedStatement
    - Parametrizzato
      - I parametri si scrivono con '?'
      - I parametri sono posizionali
    - Metodi setter per ogni tipo di dato
    - execute<OP> come in Statement

SQL Statements

- Permette di inviare istruzioni SQL al database
  - Statement
  - PreparedStatement
  - CallableStatement
    - Permette di richiamare le procedure sul db
- La risposta degli statement è un ResultSet
  - Collezione di elementi di risposta
  - Ogni riga si legge con il metodo next()
    - Restituisce un booleano
  - Possiede un metodo getter per ogni tipo
- Sia gli Statement che i ResultSet vanno chiusi!

Operazioni CRUD

- Operazioni CRUD
  - Create, Retrieve, Update, Delete

```
String insertStatement = "INSERT INTO MyTable . . .";
int ris = stmt.executeUpdate(insertStatement);
```

Inserire un nuovo record

Operazioni CRUD

```
CREATE TABLE Album ( AlbumId int, Titolo varchar(20),
Artista varchar(255), Anno int, PRIMARY KEY (AlbumId) );
```

Codice SQL per la creazione di una tabella

```
String insertStmt = "INSERT INTO ALBUM (AlbumId, Titolo,
Artista, Anno) VALUES (?, ?, ?, ?);";
try (PreparedStatement pstmt =
con.prepareStatement(insertStmt)) {
    pstmst.setInt(1, album.getAlbumId());
    pstmst.setString(3, album.getArtista());
    pstmst.setString(2, album.getTitolo());
    pstmst.setInt(4, album.getAnno());
    pstmt.executeUpdate();
    }
```

Inserire un nuovo record

Operazioni CRUD

```
String deleteStatement = String.format(
"DELETE FROM ALBUM WHERE AlbumId = %s",
album.getAlbumId());
stmt.executeUpdate(deleteStatement);
```

Rimuovere un record

```
String updateStatement = String.format(
"UPDATE ALBUM set Titolo='%s', Artista='%s', Anno=%s "
+ "WHERE AlbumId = %s", album.getArtista(),
album.getTitolo(),
album.getAnno(), album.getAlbumId());
stmt.executeUpdate(updateStatement);
```

Aggiornare un record

Codice di riferimento

```
CRUDTest.java;
```


Mappatura dei tipi

Tipo SQL	Tipo Java
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.TimeStamp

Transazioni

- Transazioni
  - `commit`
    - rende definitive tutte le modifiche apportate usando la connessione fino al precedente `commit` o `rollback`
  - `rollback`
    - annulla le modifiche fino al precedente `commit`
  - per usufruirne bisogna prima disabilitare l'*auto commit*

Transazioni

```
Connection conn = null;
try{
    conn = DriverManager.getConnection("...");
    conn.setAutoCommit(false);
    Statement st = conn.createStatement();
    st.executeUpdate("DELETE  ...");
    st.executeUpdate("INSERT ...");
     conn.commit();     }
catch (SQLException sqle)     {
       if (conn!=null)
try{
       conn.rollback();
}
catch (SQLException sqle2)
{              //log error
}
}
```

Commit e Rollback

JavaBean

- Componente riutilizzabile
- Incapsula molteplici oggetti in un singolo oggetto
- Perfetto per l'utilizzo con i database
  - Classi Java che mappano perfettamente una tabella in una classe
- Utilizzate per costruire oggetti da utilizzare per la comunicazione
  - Devono essere serializzabili
  - Il costruttore non ha argomenti
  - Possiede metodi get e set per ogni attributo

Consigli

- Non lasciare **MAI** la connessione aperta
  - Ad ogni operazione la connessione va aperta e chiusa
  - Si rischiano problemi di sicurezza
- Preferire i PreparedStatement agli Statement
  - Problemi degli Statement:
    - Difficili da leggere
    - Sensibili ad attacchi esterni (SQL Injection)
    - Le query non vengono ottimizzate
    - Comunicazione sql-java lenta
- La connessione va sempre chiusa, potrebbe essere opportuno chiuderla in un blocco *finally*

Esercizio

- Modificare BankAccount aggiungendo un database che possa memorizzare i correntisti con il loro bilancio
- Implementare le operazioni di *Inserimento correntista*
- Aggiornamento del bilancio
- Utilizzare la seguente tabella:

```
CREATE TABLE Account (
firstname varchar(50),
lastname varchar(50),
bank_number varchar(10),
balance real)
```